NRL Report 7971

A Binary Arithmetic for the Fermat Number Transform

LAWRENCE M. LEIBOWITZ

Digital Applications
Office of the Director of Research

March 18, 1976

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

UNCLASSIFIED

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 7971	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) A BINARY ARITHMETIC FOR THE FERMAT NUMBER TRANSFORM		5. TYPE OF REPORT & PERIOD COVERED Interim report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(*s*) Lawrence M. Leibowitz		8. CONTRACT OR GRANT NUMBER(*s*)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, D.C. 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS None
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE March 18, 1976
		13. NUMBER OF PAGES 16
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*)		15. SECURITY CLASS. (*of this report*) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

Binary arithmetic modulo F_t	Fast Fourier transform (FFT)
Code translation	Fermat number transform (FNT)
Convolution	Finite field
Correlation	Finite ring
Diminished-1 number representation	Modulus

(Continued)

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

The number theoretic transform (NTT) and the use of moduli of the form of the *t*th Fermat number $F_t = 2^b + 1$, $b = 2^t$, are reviewed. A binary arithmetic that permits the exact computation of the Fermat number transform (FNT) is described. This technique involves arithmetic in a binary code corresponding to the simplest one of a set of code translations from the normal binary representation of each integer in the ring of integers modulo F_t. The resulting FNT binary arithmetic operations are of the complexity of 1's complement arithmetic

(Continued)

19. Number theoretic transform (NTT)
1's complement arithmetic
Residue reduction

20. as in the case of a previously proposed technique which corresponds to another one of the set of code translations. The general multiplication of two integers modulo F_t required in the computation of FNT convolution is discussed. The exact arithmetic described here is shown to be more efficient than another previously proposed approximate arithmetic.

CONTENTS

A BINARY ARITHMETIC FOR THE FERMAT NUMBER TRANSFORM

INTRODUCTION

The discrete Fourier transform (DFT) has been defined in a finite field or ring of integers modulo an integer M by Pollard [1]. This has led to the development of number theoretic transforms (NTTs) with application to the computation of discrete convolution or correlation in a manner similar to that of the fast Fourier transform (FFT) but involving no multiplications or arithmetic roundoff error, as described in Refs. 2 through 5. The general form of the NTT is

$$X(k) = \sum_{n=0}^{N-1} x(n)\alpha^{nk} \bmod M, \qquad k = 0, 1, \ldots, N - 1,$$

where N is the sequence length, α is a root of unity of order N, and M is the modulus describing the finite field or ring of integers [1].

Many different NTTs can be defined among the various values of α, N, and M. Among these are transforms that can be computed in a highly efficient manner with α equal to 2 or a power of 2, with each multiplication replaced by a single binary word shift and an addition or subtraction. Furthermore, if N is highly composite, an FFT [6] type algorithm can be applied to improve the efficiency of the computation.

The application of NTT transforms to digital signal-processing systems was first considered by Rader [2], who proposed the use of moduli of the form of a Mersenne number $2^p - 1$, p a prime, as well as the *t*th Fermat number $F_t = 2^b + 1$, $b = 2^t$. The Mersenne number transform (MNT) leads to α of value 2, but the corresponding N values are not highly composite, thus preventing full use of the efficiencies of an FFT type algorithm. The Fermat number transform (FNT) results in an α of 2 with $N = 2b$ an integer power of 2. An FNT with $\alpha = \sqrt{2}$ and $N = 4b$ is also available, but the computation is slightly more complex.

The FNT was defined mathematically and considered in detail by Agarwal and Burrus [3]. Their disclosure includes a software realization and an analysis of FNT convolution. FFT convolution is presently in common use for discrete convolution computations, as described in Ref. 7. In their realization of the FNT, Agarwal and Burrus [3] define a binary arithmetic modulo F_t. The use of such a modulus requires b + 1 bits. The representation of the quantity $2^b = -1 \bmod F_t$ requires the (b + 1)th bit. To simplify

This report represents a part of the research performed under the Edison Memorial Fellowship in partial fulfillment of the requirements for the degree of Doctor of Science at the George Washington University School of Engineering and Applied Science.

Manuscript submitted January 2, 1976.

modular arithmetic operations on a general purpose computer of b-bit word length, Agarwal and Burrus limit their realization to b-bit arithmetic. This involves some input quantization error when -1 occurs as an input sample as well as a complete data block in error when a -1 occurs as the result of an FNT computation.

It is desirable to compute the FNT exactly. The difficulties in performing exact binary arithmetic modulo F_t become apparent when one considers, for example, multiplication or addition in a ring of integers modulo F_t involving the binary representation of -1, 2^b. For example, when $b = 4$, the product of 10000 (-1) with itself is 00001 ($+1$).

A technique for the exact computation of the FNT and its implementation are described by McClellan [8]. McClellan's approach involves the definition of a new binary code representation for the integers modulo F_t. Given a binary representation of $b + 1$ bits, $A = (a_b, a_{b-1}, \ldots, a_0)$, where A is some arbitrary integer modulo F_t, this new code is described as follows:

$$\begin{aligned} A &= 0, \quad \text{if } a_b = 1, \\ &= \sigma_{b-1}2^{b-1} + \sigma_{b-2}2^{b-2} + \ldots + \sigma_0, \quad \text{if } a_b = 0, \end{aligned}$$

where

$$\begin{aligned} \sigma_j &= 1, \quad \text{if } a_j = 1, \\ &= -1, \quad \text{if } a_j = 0. \end{aligned}$$

Thus McClellan's representation of numbers modulo F_t involves normal binary weighting with digits ± 1. This technique results in a representation of the integer 0 as 2^b in binary notation (for $b = 4$, 0 is represented as 10000) and thus the $(b + 1)$th bit is used only to represent 0. The presence of the extra bit in a multiplication or addition operation involving the number representation for 0 is detected and the normal steps in these operations are properly inhibited in order to produce a correct result. Thus binary arithmetic operations with a $(b + 1)$-bit word are effectively eliminated. McClellan shows that this number representation provides a binary arithmetic modulo F_t for negation, addition, and multiplication by integer powers of 2. This new arithmetic is shown to be similar to and as complex as ordinary 1's complement arithmetic.

In this report a binary arithmetic modulo F_t of less mathematical complexity than that of Ref. 8 will be presented. Although the translations between the quantities in the finite ring and the number representation in this binary arithmetic are trivial, the resulting binary arithmetic operations are identical to those of McClellan. Thus the FNT system design presented in Ref. 8 is applicable to this new arithmetic with the exception of the code translation, which is mathematically simpler. In addition to binary arithmetic operations involved in the computation of the FNT, this report also considers general multiplication of integers modulo F_t as required in the overall computation of FNT convolution.

CODE TRANSLATION

There are a set of code translations such that each results in the simplified binary arithmetic modulo F_t to be described here. These translations involve the one-to-one binary representation of an arbitrary number A in the ring of integers modulo F_t as the binary number corresponding to $RA - 1 \bmod F_t$, where R is any integer in the ring with an inverse. It can be shown that the code representation of McClellan [8] corresponds to the case of $R = 2^{b-1} + 1$, which for the case of $b = 4$ is $R = 9$. The simplest code translation obviously occurs for $R = 1$ for any value of b. This report will concentrate on the resulting binary arithmetic for the code translation corresponding to $R = 1$, although the discussion can be extended in a straightforward manner to any R mod F_t possessing an inverse.

DIMINISHED-1 NUMBER REPRESENTATION

To represent all integers in the ring of integers modulo F_t requires $b + 1$ bits. The additional bit is required in order to represent the number $2^b = -1 \bmod F_t$. To overcome the problem of performing binary arithmetic with this additional bit, let us describe a modified binary number system. To avoid additions and multiplications involving the additional bit, allow the additional bit to be a 1 only when the number to be represented is 0. This can be easily achieved by subtracting 1 from the normal binary representation of every integer in the ring and corresponds to the above set of code translations with $R = 1$. The normal representation and this diminished-1 representation are indicated in Table 1 for $b = 4$.

Table 1—Correspondence Between Normal and Diminished-1 Representations (b = 4)

Normal Value	Binary Representation	Diminished-1 Value
0	00000	1
1	00001	2
2	00010	3
3	00011	4
4	00100	5
5	00101	6
6	00110	7
7	00111	8
8	01000	9 (−8)
9 (−8)	01001	10 (−7)
10 (−7)	01010	11 (−6)
11 (−6)	01011	12 (−5)
12 (−5)	01100	13 (−4)
13 (−4)	01101	14 (−3)
14 (−3)	01110	15 (−2)
15 (−2)	01111	16 (−1)
16 (−1)	10000	0

In the diminished-1 number representation the b least significant bits (LSBs) indicate the value of the number. The numbers from 1 to 2^b are represented in order by the binary numbers from 0 to $2^b - 1$. With use of this representation the arithmetic operations necessary to perform convolution by Fermat number transforms (FNTs) will be considered.

BINARY ARITHMETIC OPERATIONS MODULO F_t

Negation

It can be seen from Table 1 that each of the negative numbers ($>2^{b-1}$) are the b-LSB complement of their positive counterparts. Thus with this system the negative of a nonzero number in diminished-1 representation is the complement of its b least significant bits. If the b-LSB complement is denoted by an overbar, this can be shown as

$$\begin{aligned}\overline{A - 1} &= 2^b - 1 - (A - 1)\\ &= 2^b + 1 - A - 1\\ &= (-A) - 1 .\end{aligned}$$

If the most significant bit (MSB) is 1, the negation is inhibited. An example of negation is the following:

Example 1

$$13 = -4 \bmod 17 = 0\bar{0}\bar{0}\bar{1}\bar{1} = 01100 .$$

Addition

To perform addition of two numbers represented as $A - 1$ and $B - 1$,

$$(A - 1) + (B - 1) = (A + B - 1) - 1$$

and thus

$$(A + B - 1) = [(A - 1) + (B - 1)] + 1 .$$

Since the $(b + 1)$th bit of the addends is used only to inhibit addition if an addend is 0, addition of nonzero addends involves only the b LSBs. The preceding equations indicate that a 1 must be added to the sum of two diminished-1 numbers to provide a correct result. When a carry is generated from the b-bit sum, a residue reduction modulo F_t requires the subtraction of a 1, since $2^b = -1 \bmod F_t$, and no corrective addition is necessary.

Thus to add two numbers in diminished-1 representation: If the MSB of either addend is 1, inhibit the addition, and the remaining addend is the sum. If the MSBs of both addends are 0, ignore the MSBs, add the b LSBs, complement the carry, and add it to the b LSBs of the sum. The (b + 1)th bit or MSB of the resulting sum, is the carry from the bth bit.

Examples of addition are the following:

Example 2

$$\begin{array}{lrr} \text{add} & 10000 & 0 \\ & \underline{00100} & \underline{5} \\ & 00100 & 5 \end{array}$$

Example 3

$$\begin{array}{lrrl} \text{add} & 00111 & 8 & \\ & \underline{01101} & \underline{14} & \\ & 10100 & 22 & = 5 \bmod 17 \\ & \searrow 0 & & \\ & \overline{00100} & & \end{array}$$

Example 4

$$\begin{array}{lrr} \text{add} & 00011 & 4 \\ & \underline{00101} & \underline{6} \\ & 01000 & 10 \\ & \searrow 1 & \\ & \overline{01001} & \end{array}$$

Example 5

$$\begin{array}{lrrl} \text{add} & 00100 & 5 & \\ & \underline{01011} & \underline{12} & = -5 \bmod 17 \\ & 01111 & 17 & = 0 \bmod 17 \\ & \searrow 1 & & \\ & \overline{10000} & & \end{array}$$

Code Translation

Using the preceding description of diminished-1 addition, we can present rules for the translation between a binary number B and its diminished-1 representation D.

To translate from binary to diminished-1 representation, perform a diminished-1 addition of B and the binary representation of $2^b - 1$.

Examples of translation from a binary number B and its diminished-1 representation D are the following:

Example 6

$$\begin{array}{rrl} B = & 01100 & = 12 \\ & \underline{01111} & \\ & 11011 & \\ & \searrow 0 & \\ D = & \overline{01011} & \end{array}$$

Example 7

$$\begin{array}{rrl} B = & 00000 & = 0 \\ & \underline{01111} & \\ & 01111 & \\ & \searrow 1 & \\ D = & \overline{10000} & \end{array}$$

To translate from diminished-1 to binary representation, complement the MSB of D and add it to the b LSBs.

Examples of translation from a diminished-1 representation D to a binary number B are the following:

Example 8

$$\begin{array}{rl} D = & 01011 = 12 \\ & \;\searrow\;\;\; 1 \\ \cline{2-2} B = & 01100 \end{array}$$

Example 9

$$\begin{array}{rl} D = & 10000 = 0 \\ & \;\searrow\;\;\; 0 \\ \cline{2-2} B = & 00000 \end{array}$$

Subtraction

Having defined negation and addition in the diminished-1 number system, we can perform subtraction by negating the subtrahend and adding it to the minuend, as in the following example:

Example 10

$$\begin{array}{lrl} \text{subtract} & 7 & 00110 \\ & -5 & 01011 \\ \cline{2-3} & 2 & 10001 \\ & & \;\searrow\;\;\; 0 \\ \cline{3-3} & & 00001 = 2 \end{array}$$

Multiplication by Powers of 2

In performing a multiplication, if the multiplier or multiplicand is 0, as detected by the presence of a 1 in the (b + 1)th bit, the multiplication is inhibited and the product is 0. To perform multiplication of diminished-1 numbers by powers of 2 let us consider the following:

$$(A - 1) \cdot 2 = (2A - 1) - 1$$

and thus

$$(2A - 1) = (A - 1) \cdot 2 + 1 .$$

Therefore each multiplication by 2 involves a left shift, ignoring the MSB, and a corrective addition of a 1. If the bit shifted out from the bth position is a 0, it is complemented and shifted into the LSB in order to accomplish the addition of a 1. If this bit is a 1, a subtraction of 1 is also required to accomplish a residue reduction. With the corrective addition of +1, these cancel out and a 0 is shifted into the LSB. Thus for each factor of 2 a left circular shift of the b LSBs is required and the bit circulated into the LSB is complemented. For negative powers of 2 a right circular shift is performed with bits circulated to the bth bit (second MSB) complemented. (Multiplication by negative powers of 2 can also be performed using a positive power of 2 followed by a negation. (For example, $2^{-k} = 2^{N-k} = 2^b \cdot 2^{N-b-k} = -2^{b-k}$, where N and b are as previously defined and $k < b$.)

An example of multiplication by powers of 2 is the following:

Example 11

$$8 \times 11 = 2^3 \times 11 = 2 \times 2 \times 2 \times 11 = 88 = 3 \bmod 17$$

11	01010
2×11	00100
4×11	01001
8×11	00010 = 3 mod 17

To realize the full efficiency of the FNT, it is necessary to accomplish multiplication by powers of 2 by a single multibit shift. The preceding procedure can be extended in a straightforward manner to provide such multiplication.

General Multiplication

The last operation required to carry out convolution with the Fermat number transform (FNT) is a general multiplication by any two integers modulo F_t. A multiplication of the numbers A and B represented as A − 1 and B − 1 in the diminished-1 number system is performed as

$$\begin{aligned}(A-1)(B-1) &= A \cdot B - (A+B) + 1 \\ &= (A \cdot B - 1) - (A + B - 1) + 1,\end{aligned}$$

and the desired result is

$$(A \cdot B - 1) = (A-1)(B-1) + (A+B-1) - 1.$$

Thus, to carry out such an operation, ignore the MSB, perform a binary multiplication of the diminished-1 representations of A and B, add this result to the b LSBs of the diminished-1 addition of A and B, and then perform a residue reduction by a diminished-1 subtraction of the b-MSBs from the b-LSBs. (This particular general multiplication scheme is not applicable with code translations other than that corresponding to R = +1.) As discussed previously, if the MSB of either multiplier or multiplicand is 1, the multiplication is inhibited and the result is set to 0.

An example of general multiplication is the following:

Example 12

```
multiply           15
                  X10
                  ---
                  150 = 14 mod 17

binary multiply         01110        diminished-1 add    01110
                        01001                            01001
                        -----                            -----
                         1110                            10111
                      0111000                            \
                      --------                            \
                      01111110                             ↘0
                                                         -----
binary add              00111 <------------------------- 00111
                      --------
                      10000101
                      \__/
                        \
                         ↘
                         ___
residue reduction        0111
                        -----
                        01100
                        \
                         ↘1
                        -----
                        01101 = 14 mod 17
```

An alternate multiplication technique can also be considered. Assuming that the numbers are determined to be nonzero and a multiplication is required, a translation to normal binary coding is completed. Following a binary multiplication, a residue reduction by diminished-1 subtraction of the b MSBs of the product from the b LSBs is performed. The result is the desired product. This is a realization of the equation (A • B − 1) = A • B − 1.

An example of this alternate technique of general multiplication is the following:

Example 13

```
multiply           15
                  X10
                  ---
                  150 = 14 mod 17

translate               01110        binary multiply    ↗ 01111
                        \                              ↗  01010
                         ↘1                                -----
                        -----                              11110
                        01111 ----------------------/     11110
                                                          --------
translate               01001                             10010110
                        \                                 \__/
                         ↘1                                  ↘
                        -----        residue reduction     ___
                        01010 ---------------------/       0110
                                                          -----
                                                          01100
                                                          \
                                                           ↘1
                                                          -----
                                                          01101 = 14 mod 17
```

Another general multiplication technique involves a translation of either the multiplier or the multiplicand to normal binary form, the other remaining in diminished-1 representation. The operations involved in comparison to the previous techniques include a diminished-1 to binary translation and a general diminished-1 addition.

In comparing these general multiplication schemes, several factors should be considered. The basic tradeoff involves the general diminished 1 and binary additions as compared to the translation of numbers in diminished-1 representation to normal binary representation. The remaining operations of binary multiplication, residue reduction in diminished-1 notation, and multiplication by 0 are identical in the various techniques. Since in most cases the translation from diminished-1 to binary will be simpler and faster than a general binary or diminished-1 addition, the second technique is preferable to the others.

COMPARISON OF DIMINISHED-1 AND NORMAL-BINARY EFFICIENCY

The preceding diminished-1 procedures accomplish exact arithmetic modulo F_t by a code translation that represents 0 by the $(b + 1)$th bit with arithmetic operations that treat 0 as a special case. It is thus logical to consider a set of procedures that operate on normal binary representations with special rules for handling -1 as indicated by the $(b + 1)$th bit. An arithmetic using such procedures to compute the FNT exactly would be an extension of the b-bit approximate arithmetic modulo F_t of Agarwal and Burrus [3] described by the following rules:

- *Negation*—Complement each bit and add 2 to the result.
- *Addition*—Add the two b-bit integers and end-around subtract any carry bit.
- *Subtraction*—Negate the subtrahend and add the result to the minuend.
- *General multiplication*—Multiply and get $C_L + C_H 2^b$, where C_L and C_H are words formed from the b least and most significant bits respectively, and from C_L subtract C_H mod F_t.
- *Multiplication by a power of 2*—To multiply by 2^k, load the data in the lower half of a 2b-bit double register, shift left k positions, and from C_L subtract C_H mod F_t. To multiply by 2^{-k}, load the data in the upper half of a 2b-bit double register, shift right k positions, and from C_H subtract C_L mod F_t.

Any extension of these rules in order to accomplish the exact FNT computation using $b + 1$ bits can only be less efficient with respect to the number of operations required. Thus if it can be shown that $b + 1$ bit diminished-1 exact arithmetic modulo F_t is more efficient than the b-bit approximate arithmetic modulo F_t of Agarwal and Burrus [3], it can be assumed to be even more efficient than any extension of the latter.

To compare the arithmetics, the number of basic binary arithmetic operations (additions, multiplications, shifts) involved in computing an N-point FNT convolution will be considered for each. An FNT convolution computation is assumed to consist of two FNTs followed by N general multiplications, N divisions by N, and another FNT. Additionally the diminished-1 computation requires N encode and decode operations. Any sequence reordering procedures required in an actual computation of FNT convolution do not depend on the arithmetic used and are thus not considered here. It is assumed that the complexity of a subtraction is equivalent to that of an addition and that complementation can be neglected.

Considering the b-bit approximate arithmetic first, the complexity of arithmetic modulo F_t in terms of basic binary operations is indicated in Table 2. There are $(N/2) \log_2 N$ butterfly operations in any of the standard FFT algorithms. Since each butterfly consists of an addition, subtraction, and multiplication by 2^k, each FNT requires

$5N \log_2 N$ binary additions

and

$\frac{N}{2} \log_2 N$ binary shifts.

In computing the convolution there are N general multiplications requiring

N binary multiplications

and

4N binary additions,

and there are N divisions by N requiring

N binary shifts

and

4N binary additions.

The total number of basic binary operations for FNT convolution with b-bit arithmetic is then

$15N \log_2 N + 8N$ binary additions,

$\frac{3N}{2} \log_2 N + N$ binary shifts,

and

N binary multiplications.

Table 2—Basic Binary Operations for FNT Convolution Using b-Bit Arithmetic

Mod F_t Operation	Basic Binary Operation	Number of Operations
Addition	Addition	2
Subtraction	Addition	4
Multiplication by 2^k	Shift	1
	Addition	4
General multiplication	Multiplication	1
	Addition	4

For the (b + 1)-bit diminished-1 exact arithmetic the complexity of arithmetic modulo F_t in terms of basic binary operations is indicated in Table 3. Thus there are

$5N$ binary additions

involved in the code translations. Each FNT requires

$2N \log_2 N$ binary additions

and

$\frac{N}{2} \log_2 N$ binary shifts.

The general multiplication consists of

$4N$ binary additions

and

N binary multiplications,

and division by N requires

N binary shifts.

The total number of basic binary operations for FNT convolution using b + 1 bit diminished-1 arithmetic is thus

$6N \log_2 N + 9N$ binary additions,

$\frac{3N}{2} \log_2 N + N$ binary shifts*,

and

N binary multiplications.

Table 3—Basic Binary Operations for FNT Convolution Using (b + 1)-Bit Diminished-1 Arithmetic

Mod F_t Operation	Basic Binary Operation	Number of Operations
Encode	Addition	2
Decode	Addition	1
Addition	Addition	2
Subtraction	Addition	2
Multiplication by 2^k	Shift	1
General multiplication	Multiplication Addition	1 4

*The diminished-1 multiplication by 2^k requires only b-bit registers, whereas the other procedure requires 2b-bit registers.

The total number of shifts and multiplications is the same for both arithmetic modulo F_t procedures, but the number of additions is significantly less for diminished-1 arithmetic. Let K be the ratio of b-bit normal binary to (b + 1)-bit diminished-1 total additions. Then

$$K = \frac{15 \log_2 N + 8}{6 \log_2 N + 9},$$

with the values of K indicated in Table 4 for various values of N. The number of additions required for computation of FNT convolution using b-bit approximate arithmetic is thus about twice that required using (b + 1)-bit diminished-1 exact arithmetic. For a typical hardware multiplication time of about 5 times that of an addition and with increasing $N \geqslant 16$, the additions become the most significant portion of the FNT convolution computation. Under these conditions K is an approximate measure of the relative complexity of the two arithmetics. Thus any extension of the b-bit approximate binary arithmetic modulo F_t of Agarwal and Burrus [3] to (b + 1)-bit exact arithmetic will be significantly less efficient than the (b + 1)-bit diminished-1 exact binary arithmetic presented in this report. If it is desired to limit binary arithmetic to b bits with resulting approximation, it is more efficient to use diminished-1 arithmetic without a representation for 0 mod F_t than the arithmetic of Agarwal and Burrus.

CONCLUSION

A binary arithmetic applicable to the exact computation of the Fermat number transform has been presented. This arithmetic involves operations on a binary representation of the integers modulo F_t diminished by 1. This diminished-1 representation is mathematically simpler than that of McClellan [8], both being among a general set of such binary code translations. This arithmetic provides simple realizations of all the operations required to compute the Fermat number transform. These operations are identical to those described and implemented by McClellan [8] and are of the complexity of 1's complement arithmetic. The general multiplication of two integers modulo F_t in diminished-1 representation can be accomplished in a straightforward manner. The diminished-1 exact arithmetic is shown to be more efficient than the b-bit approximate arithmetic of Agarwal and Burrus [3] and thus more efficient than any extension of it to (b + 1)-bit exact arithmetic.

Table 4—Ratio of the Number of Basic Binary Additions Required for FNT Convolution Using b-Bit Arithmetic to that of (b + 1)-Bit Diminished-1 Arithmetic for Various Values of N

N	K	N	K
8	1.96	64	2.18
16	2.06	128	2.22
32	2.13	∞	2.5

ACKNOWLEDGMENTS

The author expresses his appreciation to Professor J. H. McClellan of the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, for his comments and suggestions. The work of Mrs. A. Latham and Mrs. A. Sullivan in the preparation of the manuscript is greatly appreciated.

REFERENCES

1. J.M. Pollard, "The Fast Fourier Transform in a Finite Field," *Mathematics of Computation* 25, 365-374 (Apr. 1971).
2. C.M. Rader, "Discrete Convolutions via Mersenne Transforms," *IEEE Transactions on Computers* C-21, 1269-1273 (Dec. 1972).
3. R.C. Agarwal and C.S. Burrus, "Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering," *IEEE Transactions on Acoustics, Speech and Signal Processing* ASSP-22, 87-97 (Apr. 1974).
4. C.M. Rader, "Convolution and Correlation Using Number Theoretic Transforms," Section 6.19 of *Theory and Application of Digital Signal Processing*, L.R. Rabiner and B. Gold, Prentice-Hall, Englewood Cliffs, N.J., 1975.
5. L.M. Leibowitz, "Fast Convolution by Number Theoretic Transforms," NRL Report 7924, Sept. 12, 1975.
6. J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation* 19, 297-301 (Apr. 1965).
7. T.G. Stockham, Jr., "High-Speed Convolution and Correlation," AFIPS Proceedings, 1966 Spring Joint Computer Conference, Vol. 28, pp. 229-233, Spartan, Washington, D.C., 1966.
8. J.H. McClellan, "Hardware for the Fermat Number Transform," Technical Note ESD-TR-75-149, Lincoln Laboratory, M.I.T., Apr. 1975.